**과목명: 시스템프로그래밍**

**2분반**

**<<Project #1>>**

**서강대학교 [컴퓨터공학과]**

**[20171666]**

**[이예은]**

목 차

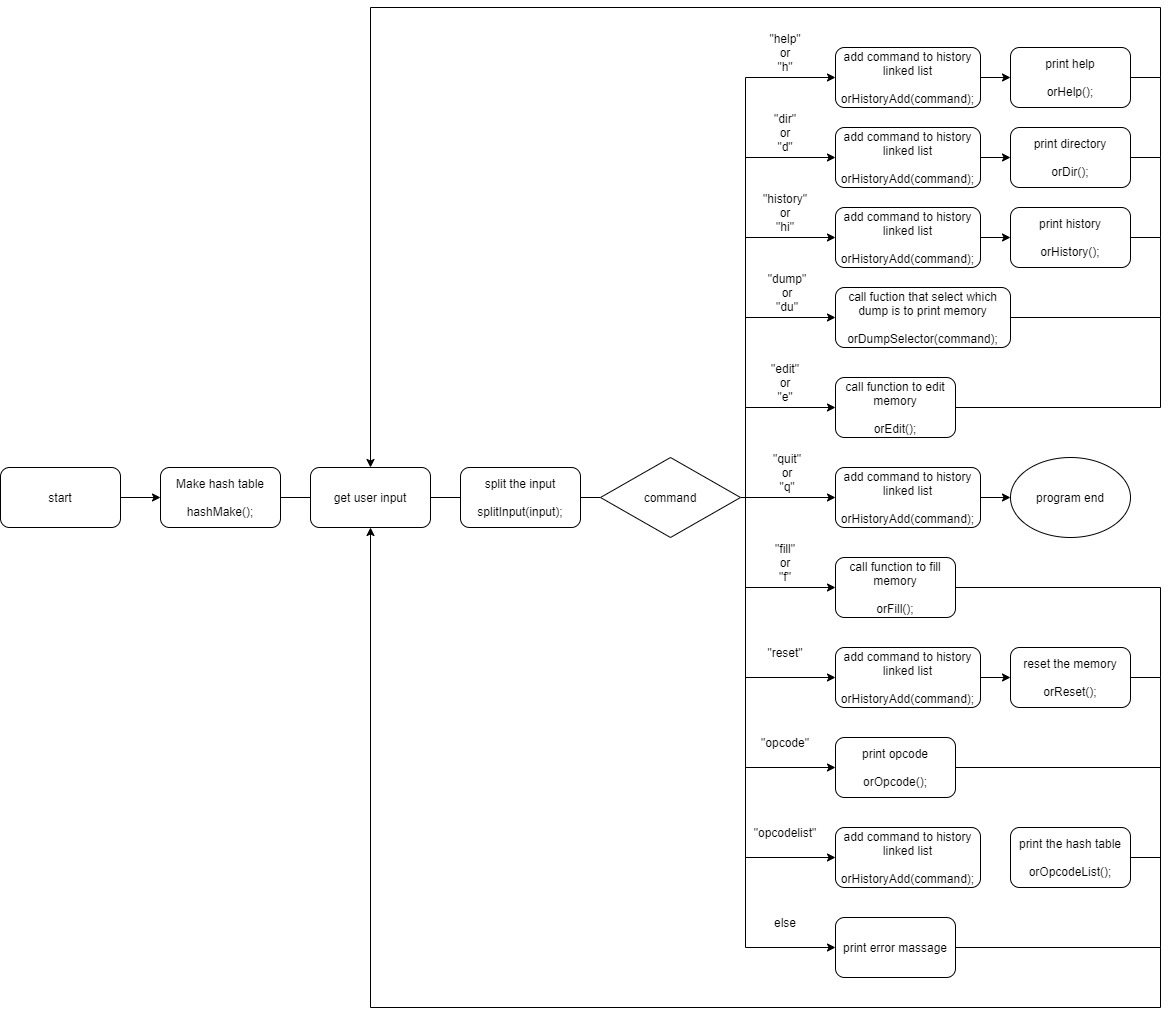
1. **프로그램 개요**
2. **프로그램 설명**
   1. 프로그램 흐름도
3. **모듈 정의**
   1. 정의한 모듈에 대해 간략적인 설명
4. **전역 변수 정의**
5. **코드 설명**
6. **프로그램 개요**

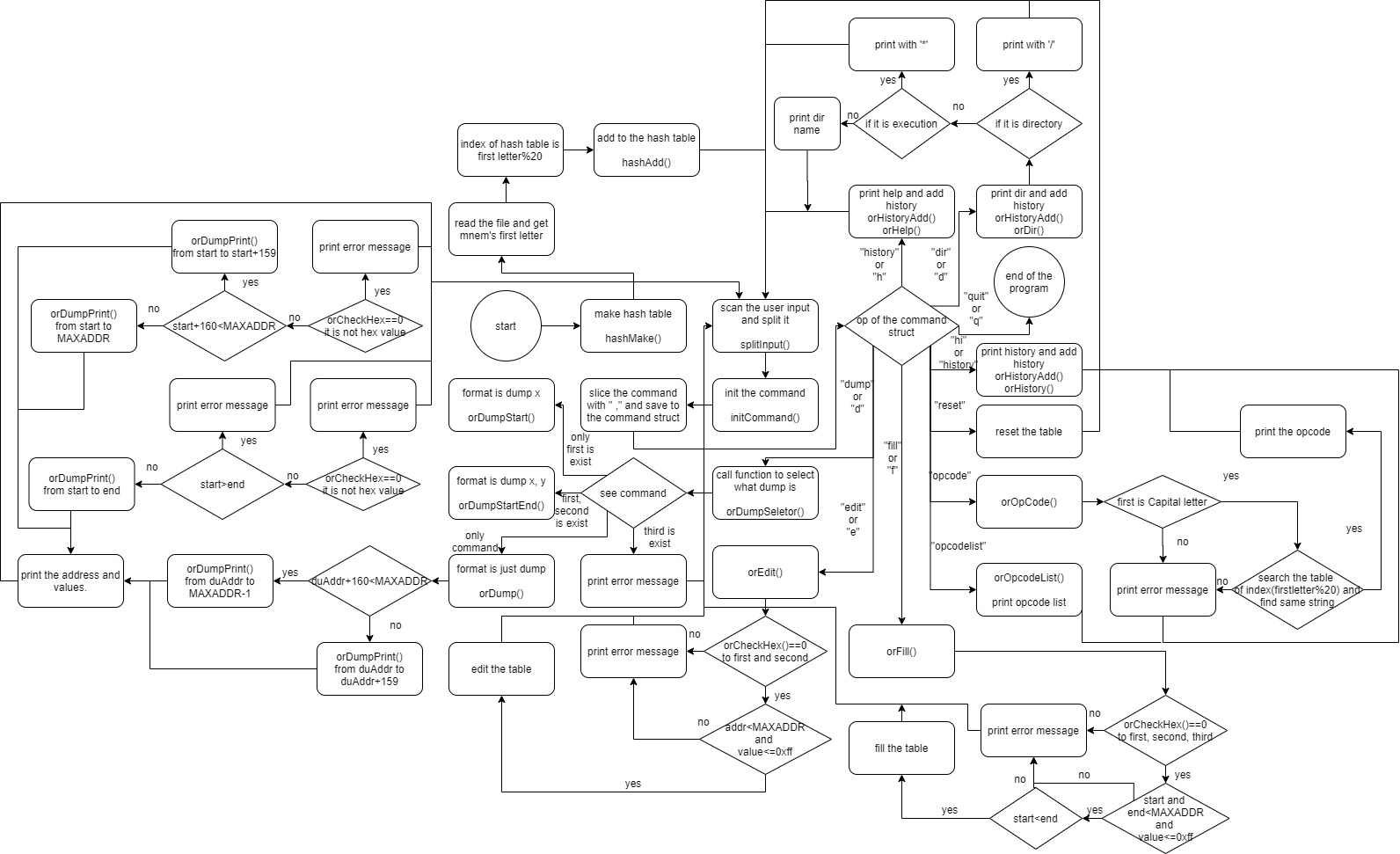
이 프로그램은 유사한 입력 프롬프트로 링크, 로더 등을 실행할 쉘과 object 코드가 들어갈 메모리공간, mnemonic을 opcode로 바꾸는 opcode 테이블을 구현하고 이와 관련된 각종 명령어들을 수행하는 프로그램입니다. 기능은 크게 쉘 관련 명령어 help, dir, quit, history와 메모리 관련 명령어 dump, fill, edit, reset이 있고, opcode 테이블 관련 opcode mnemonic, opcodelist 명령어가 있습니다. 사용자가 명령어를 입력하면 그에 따른 결과를 화면에 출력하는 방식으로 구현된 프로그램입니다.

1. **프로그램 설명**

프로그램의 전반적인 동작은 다음과 같습니다. 먼저 실행 하자마자 history를 저장할 linked list와 hash table을 구현합니다. 이후 사용자의 입력을 받아들여 알맞게 분리한 후 명령어에 맞는 함수들을 실행시킵니다. 함수들의 실행이 끝나고 에러 메세지 혹은 올바른 결과가 출력되었다면 다시 사용자의 입력 받는 부분으로 돌아와 사용자가 quit 명령어를 입력하기 전까지 위 과정을 반복합니다.

* 1. **프로그램 흐름도**

1. **Main 함수 흐름**
2. **전체적인 프로그램 흐름도**



1. **모듈 정의**

구현한 함수들은 다음과 같습니다.

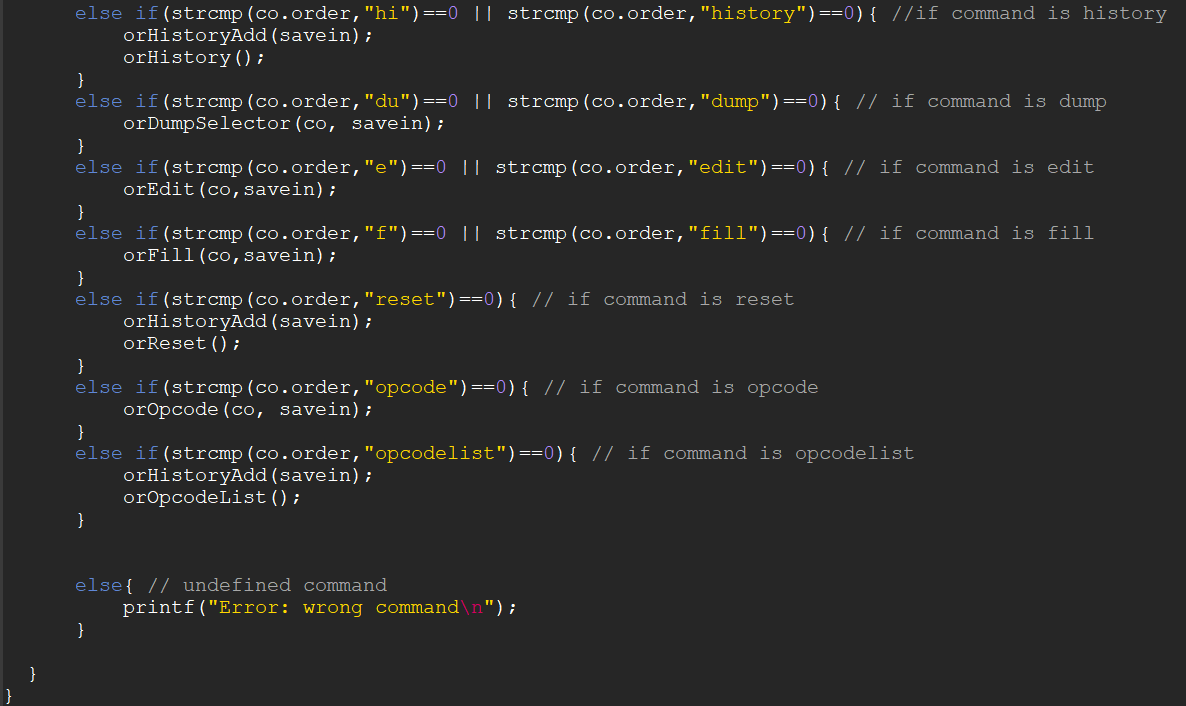
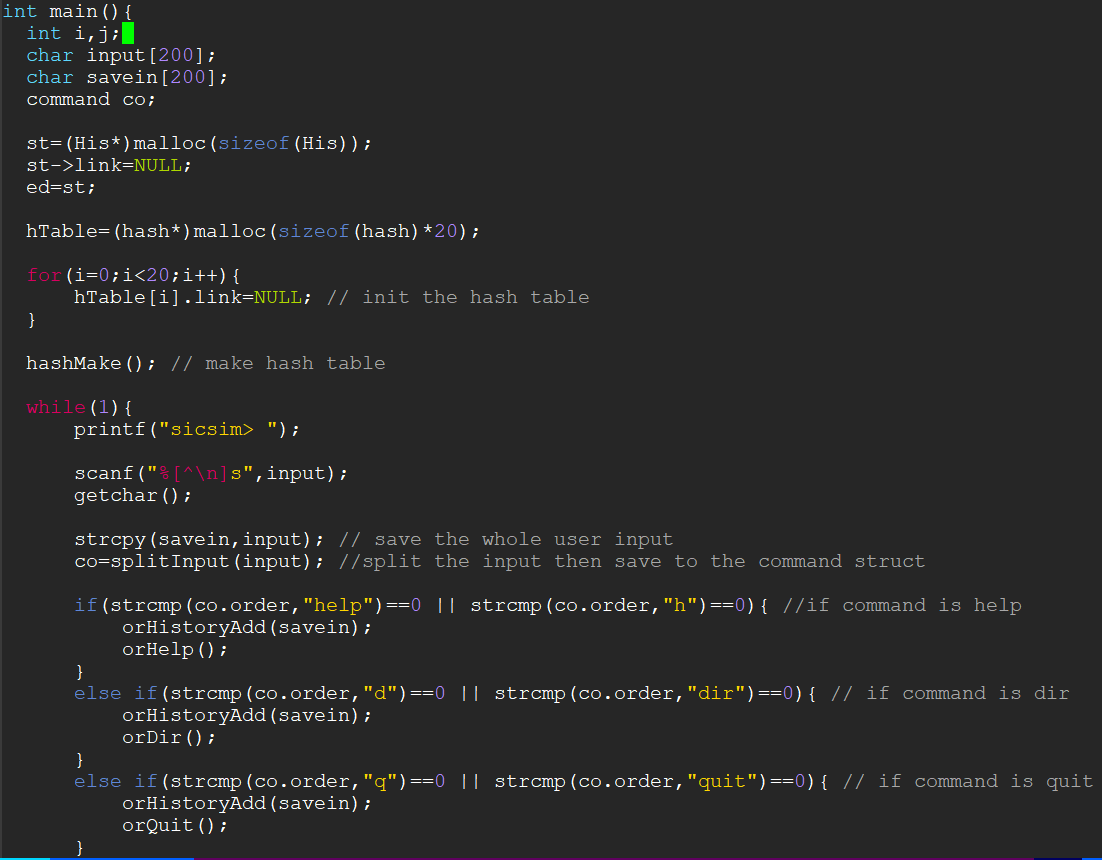
|  |  |
| --- | --- |
| int main() | History list 및 hash table을 만들고 초기화 한 후 유저의 입력을 받습니다. 유저의 입력을 splitInput 함수로 보내 쪼갠 후 명령어부분에 따라 함수를 불러옵니다. 잘못된 명령어라면 에러메세지를 출력합니다. 모든 출력을 마친 후 다시 입력받는 부분으로 돌아와 반복합니다. |
| Command splitInput(char[]) | 유저의 입력을 인자로 받아 “ “와 “,”를 기준으로 문자열을 자릅니다. 이후 command 구조체에 저장, 반환합니다. |
| Command initCommand() | Command 구조체를 “-“ 값으로 초기화합니다. |
| Void orHelp() | shell에서 실행 가능한 명령어들을 출력합니다. |
| Void orDir() | 현재위치의 파일들을 종류에 맞게 출력합니다. 디렉토리일 경우 ‘/’를 붙이고, 실행파일일 경우 ‘\*’를 붙여 출력합니다. |
| Void orQuit() | 프로그램을 끝냅니다. |
| Void orHistory() | History list를 돌며 history를 출력합니다. |
| Void orHistoryAdd(char[]) | 입력을 인자로 받아 새로운 history struct를 만들어 list에 추가합니다. |
| Void orDump(command, char[]) | Command 구조체와 유저입력을 인자로 받아 history list에 넣은 후 전역변수 duAddr부터 메모리 10줄을 출력합니다. 메모리 10줄이 최대 메모리를 초과할 경우 초과하지 않는 범위만큼만 출력 후 duAddr을 처음으로 되돌립니다. |
| Void orDumpStart(command, char[]) | Command 구조체와 유저입력을 인자로 받아 잘못된 명령어는 에러 메세지를 출력합니다. 올바른 명령이라면 history list에 추가하고 start 부터 160개의 메모리를 출력합니다. 160개가 최대 메모리를 초과할 경우 초과하지 않는 범위까지 출력합니다. |
| Void orDumpStartEnd(command, char[]) | Command 구조체와 유저입력을 인자로 받아 잘못된 명령어는 에러 메세지를 출력합니다. 올바른 명령이라면 history list에 추가하고 start부터 end 까지 메모리를 출력합니다. 최대 메모리를 넘거나 end가 start보다 작은 경우 에러 메세지를 출력합니다. |
| Void orDumpSelector(command, char[]) | Command 구조체와 유저입력을 인자로 받아 Dump의 명령어가 어느 종류인지 판단합니다. Dump만 온경우, dump x의 형식으로 온 경우, dump x, y의 형식으로 온 경우에 따라 함수를 부르고 형식이 맞지 않는다면 에러 메세지를 출력합니다. |
| Void orDumpPrint(int, int) | start주소와 end 주소를 인자로 받아 start부터 end까지 메모리를 형식에 맞게 출력합니다. |
| Int orCheckHex(char[]) | 문자열을 인자로 받아 그 문자열이 hex의 표현범위, 0-9, A-Z 안에 있는지 확인 후 hex 표현이 맞다면 1을, 아니라면 0을 반환합니다. |
| Void orEdit(command, char[]) | Command 구조체와 유저입력을 인자로 받아 addr의 값을 value로 바꿉니다. 잘못된 형식이나 범위를 넘어간 addr, value가 들어올 경우 에러 메시지를 출력합니다. |
| Void orFill(command, char[]) | Command 구조체와 유저입력을 인자로 받아 start부터 end까지의 값을 value로 바꿉니다. 잘못된 형식이거나 범위를 넘어간 start, end, value값, end가 start보다 작은 경우 에러 메세지를 출력합니다. |
| Void orReset() | 메모리 배열을 0으로 리셋합니다. |
| Void hashMake() | Opcode.txt파일을 받아 알맞은 hash key에 따라 hash table에 추가해줍니다. |
| Void hashAdd(char[], char[], char[], int) | Opcode, mnemonic, format, hash key를 인자로 받아 새로운 hash 구조체를 만들고 hash table에 추가해줍니다. |
| Void orOpcode(command, char[]) | Command 구조체와 유저입력을 인자로 받아 hash table에서 mnemonic에 맞는 opcode를 출력해줍니다. 잘못된 형식이거나 mnemonic이 table에 없다면 에러 메시지를 출력합니다. |
| Void orOpcodeList() | 전체 opcode list를 출력해줍니다. |

1. **전역변수 정의**

사용된 전역변수는 다음과 같습니다.

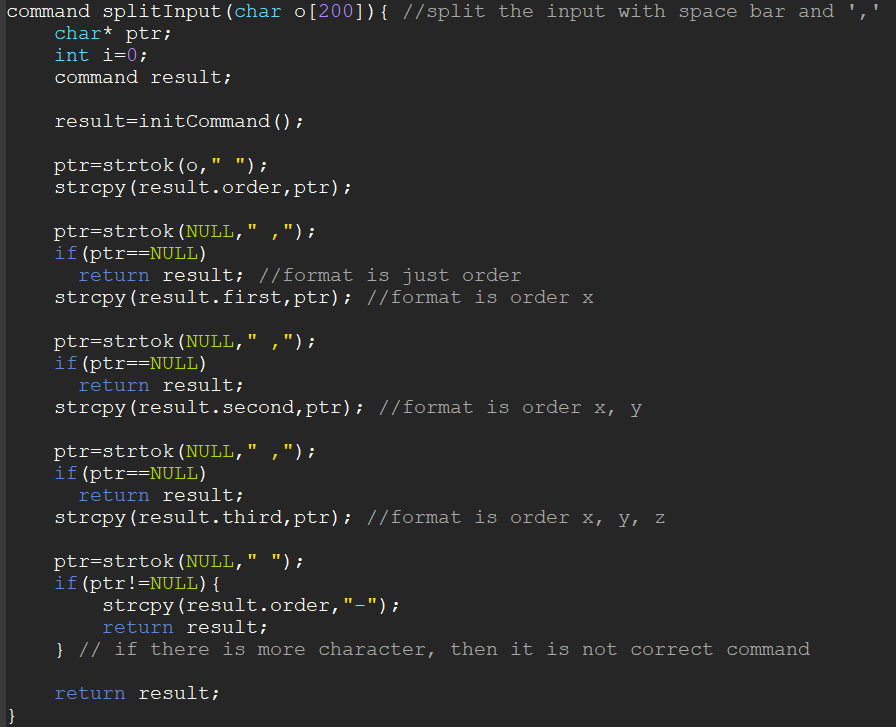
|  |  |
| --- | --- |
| Const int MAXADDR | 최대 메모리 주소를 저장하고 있는 상수 |
| Const int MAXVALUE | 메모리에 저장될 수 있는 최대 값을 저장하고 있는 상수 |
| His | History list를 위한 구조체. 내부에 유저입력을 저장하는 order[200]와 다음 list를 가리키는 포인터가 존재. |
| Command | 유저 입력을 쪼개 저장하기 위한 구조체. 내부에 명령어 저장하는 order[200]과 다음 인자들을 저장하는 first[10], second[10], third[10]이 존재 |
| Hash | Hash table을 위한 구조체. 내부에 Opcode 저장하는 op[100], mnemonic 저장하는 mnem[20], format을 저장하는 form[10]이 존재 |
| His\* st | History list의 처음을 가리키는 포인터 |
| His\* ed | History list의 끝을 가리키는 포인터 |
| Unsigned char memory[65536][16] | 메모리 명령어를 위한 메모리 1Mbyte의 정적할당 배열 |
| Int duAddr | Dump 명령어를 위한 주소 인덱스 저장하는 integer |
| Hash\* hTable | Hash table을 가리키는 포인터 |

1. **코드 설명**
   1. **Main**



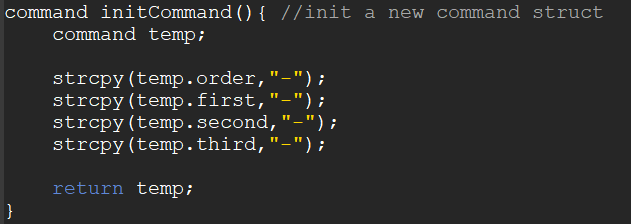
우선 history와 hash table에 필요한 전역변수들을 할당하고 초기화시켜줍니다. 이후 hashMake를 불러 hash table을 만들어주고 유저의 입력을 받는 부분으로 넘어갑니다. 유저의 입력을 받은 후 savein에 전체 입력을 백업하고 splitInput을 불러 co에 자른 입력이 저장된 command 객체를 넣습니다. 이후 명령어인 co.order를 보고 어떤 명령어인지 판단하여 그에 맞는 or 함수들을 불러주고 다시 입력으로 돌아옵니다. 만약 어떤 명령어와도 맞지 않으면 에러 메시지를 출력한 후 다시 입력으로 돌아옵니다. 입력으로 quit을 받을 때 까지 반복합니다.

* 1. **splitInput**



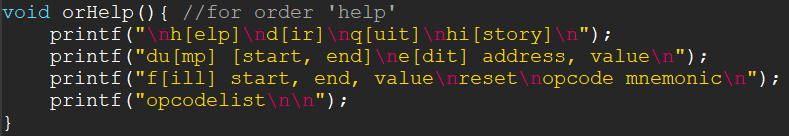
Result에 새로 초기화 된 command 구조체를 넣고 인자로 받은 유저의 입력을 우선 “ “를 기준으로 쪼갭니다. 맨 처음 쪼개진 부분이 명령어이므로 order에 넣고, 나머지부분은 “ “나 “,”를 기준으로 쪼갭니다. 명령어 이후에 다른 인자들이 입력되었다면 차례로 first, second, third로 들어갑니다. 없다면 초기화 된 상태 “-“가 유지됩니다. 이후 만들어진 command 구조체를 반환합니다.

* 1. **initCommand**



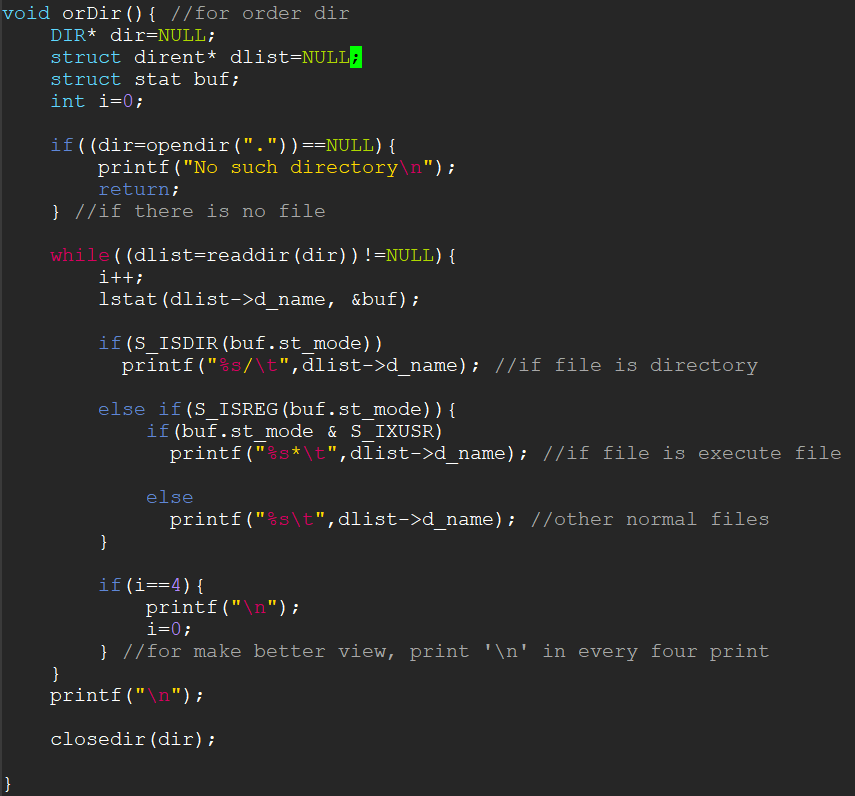
새로운 command 구조체를 만들어 내부 요소를 전부 “-“로 초기화 시킵니다. 이후 만들어진 command 구조체를 반환합니다.

* 1. **orHelp**



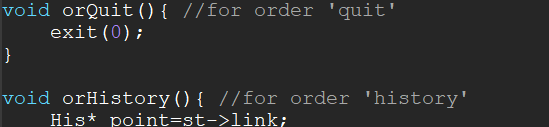
도움말을 출력해줍니다.

* 1. **orDir**



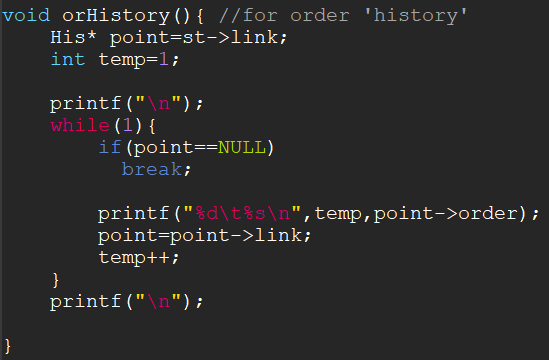
디렉토리 구조체 dir을 만든 후 opendir로 현재위치를 열어 dir에 저장해줍니다. 만약 dir이 NULL일 경우 올바르게 열지 않은 것이므로 에러 메시지를 출력합니다. 이제 이 dir을 readdir로 읽어들여 파일 이름을 출력합니다. Dirent.h의 메크로에 따라 S\_ISDIR()이 true로 반환된다면 디렉토리 파일인 것이므로 끝에 ‘/’을 추가하여 출력합니다. S\_ISREG()로 다른 파일이고, S\_IXUR로 실행파일임이 판명난다면 끝에 ‘\*’를 추가하여 출력합니다. 나머지는 일반 파일이므로 그냥 출력합니다. 또한 보기 편하게 하기 위해 i라는 플래그를 두어 파일이름 4개 출력한 후 엔터를 출력하도록 구현했습니다.

* 1. **orQuit**



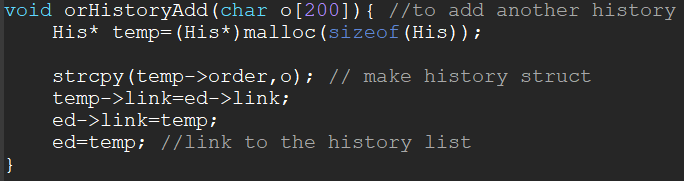
프로그램을 종료합니다.

* 1. **orHistory**



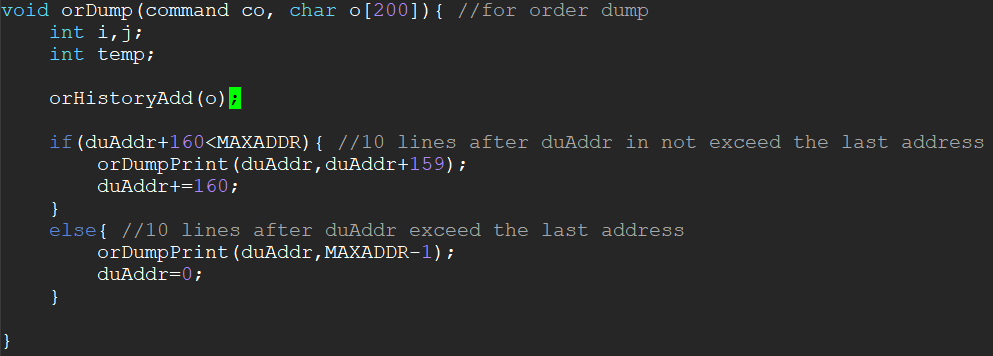
History list의 현재위치를 나타내는 point 포인터를 하나 선언하고 초기 위치로 st->link를 넣어줍니다. Point가 NULL이 되면 list를 모두 돌았으므로 break하고 아니라면 저장된 order를 출력 후 point를 다음 link로 옮겨줍니다.

* 1. **orHistoryAdd**



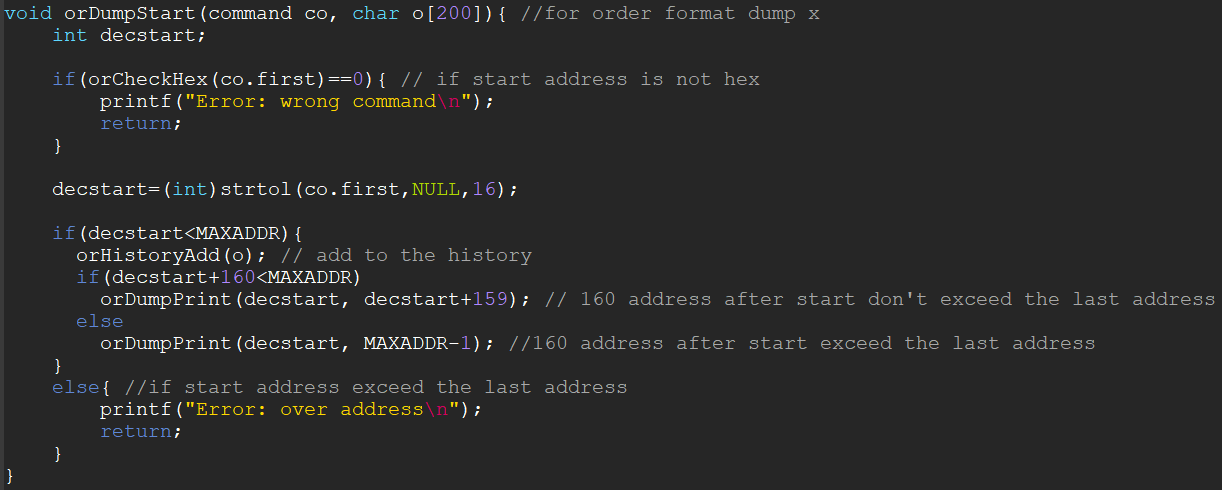
새로운 his 구조체 temp를 만들고 내부에 명령어 o[200]을 넣어줍니다. 이후 끝을 가리키는 포인터 ed 뒤에 연결해줍니다.

* 1. **orDump**



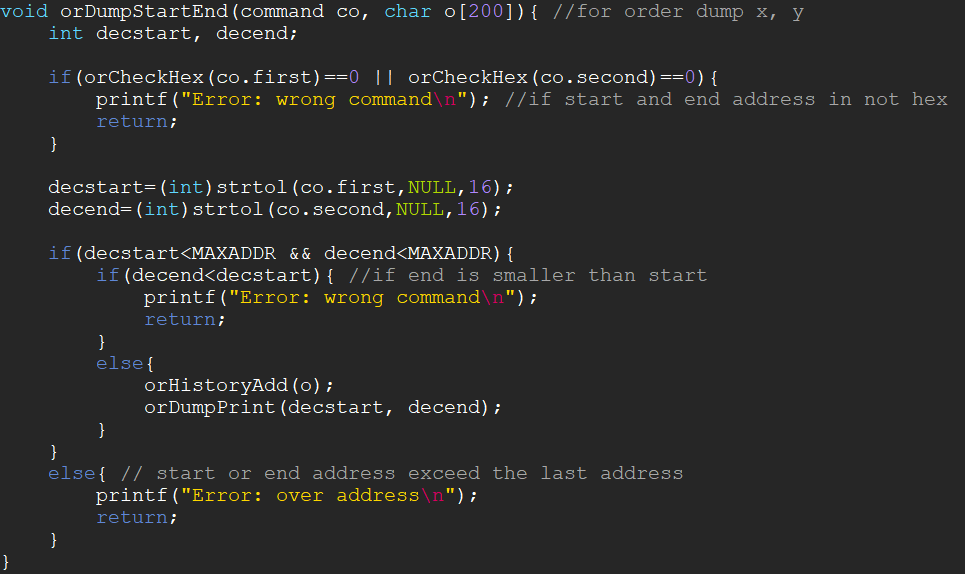
먼저 orHistoryAdd를 불러 history를 추가해줍니다. 이후 dump가 출력해야할 주소를 갖고 있는 duAddr로 10줄을 출력했을 때(160개를 출력했을 때) MAXADDR을 넘지 않는다면 orDumpPrint(duAddr, duAddr+159)를 불러 duAddr부터 10줄(160개)를 출력해줍니다. 이후 duAddr에 160을 더해 다음 출력해야할 주소를 가지도록 합니다.만약 duAddr로 10줄을 출력했을 때, MAXADDR을 넘는다면 최대 주소까지만 출력하도록 orDumpPrint(duAddr, MAXADDR-1)을 불러줍니다. 이후 duAddr을 0으로 초기화해줍니다.

* 1. **orDumpStart**



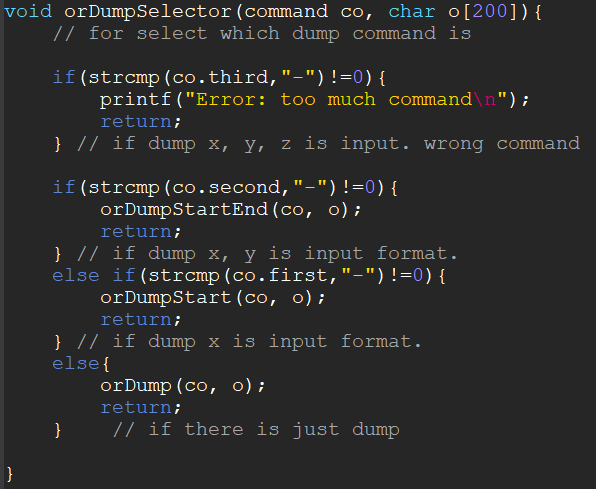
우선 입력받은 start의 값인 co.first가 16진수의 형식인지 확인하기 위해 orCheckHex를 불러줍니다. hex값이 아니라면 에러 메시지를 출력하고 함수를 종료합니다. hex값인 것이 확인됐다면 decstart에 이 값을 10진수 integer로 넣어주고 MAXADDR보다 큰지 검사합니다. 크다면 최대주소를 넘어간 것이므로 에러 메시지를 출력하고 함수를 종료합니다. 아니라면 올바른 입력이므로 orHistoryAdd로 history에 넣어주고 decstart에서 10줄(160개)를 출력했을 때 MAXADDR을 넘는지 확인합니다. 넘는다면 최대까지만 출력하도록 orDumpPrint(decstart, MAXADDR-1)을 불러주고 아니라면 160개를 출력하도록 orDumpPrint(decstart, decstart+159)를 불러줍니다.

* 1. **orDumpStartEnd**



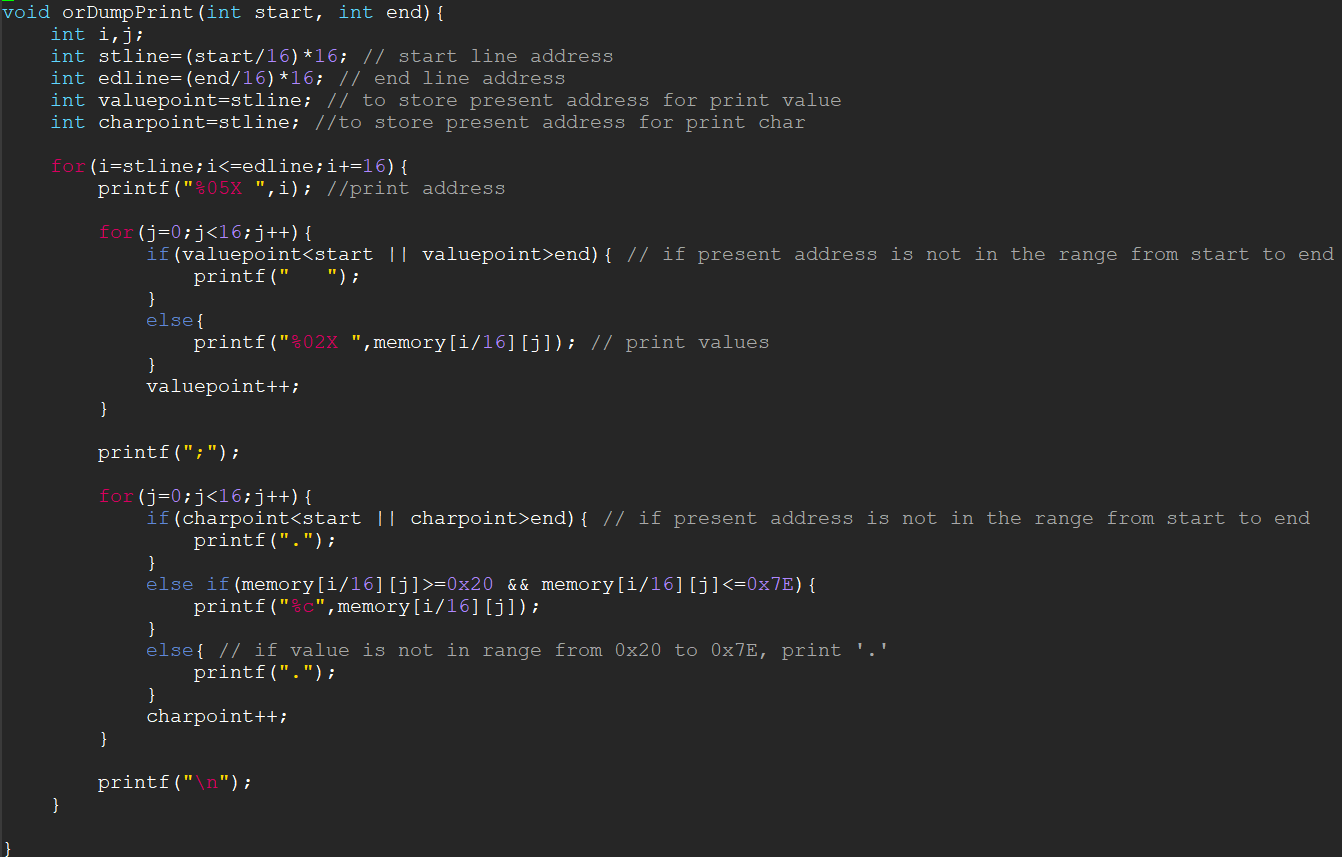
먼저 Start, end 값인 co.first와 co.second가 hex값인지 확인합니다. 둘 중 하나라도 아니라면 에러 메시지를 출력하고 함수를 종료합니다. 둘 다 hex값이 맞다면 decstart와 decend에 각각 10진수 integer로 변환된 값을 넣어줍니다. 그 후 decstart와 decend가 모두 MAXADDR보다 작은지 확인합니다. 만약 둘 중 하나라도 크다면 메모리 범위를 넘어간 것이므로 에러 메시지를 출력하고 함수를 종료합니다. 아니라면 decstart가 decend보다 큰지 검사합니다. 크다면 잘못된 명령이므로 에러 메시지를 출력하고 함수를 종료합니다. 작다면 올바른 명령이므로 orHistoryAdd로 history를 추가하고 orDumpPrint(decstart, decend)로 start부터 end까지 출력합니다.

* 1. **orDumpSelector**



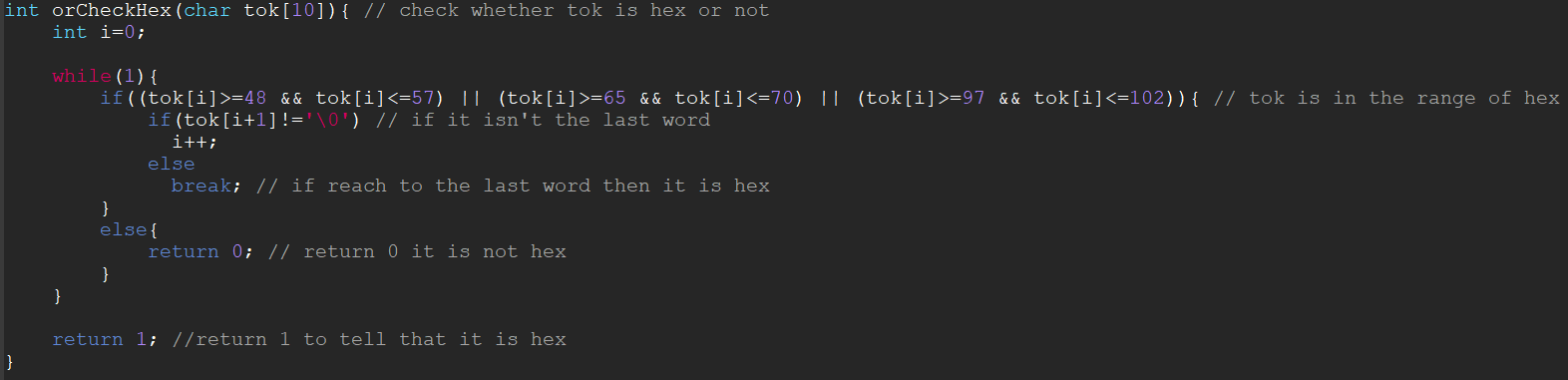
어떤 형식의 dump 명령인지 판별합니다. 만약 쪼개진 명령을 갖고있는 co에서 co.third가 “-“가 아니라면 세번째 인자로 다른 값이 들어왔단 것으로 잘못된 명령으로 인식해 에러 메시지를 출력하고 함수를 종료합니다. Co.second가 “-“가 아니라면 두번째 인자가 들어왔단 뜻이므로 형식은 dump start, end가 되어 orDumpStartEnd를 불러줍니다. Co.first가 “-“라면 첫번째 인자만 들어왔단 뜻이므로 형식은 dump start가 되어 orDumpStart를 불러줍니다. 만약 아무런 인자 없이 명령어 order만 있다면 dump만 들어왔단 뜻이므로 orDump를 불러줍니다.

* 1. **orDumpPrint**



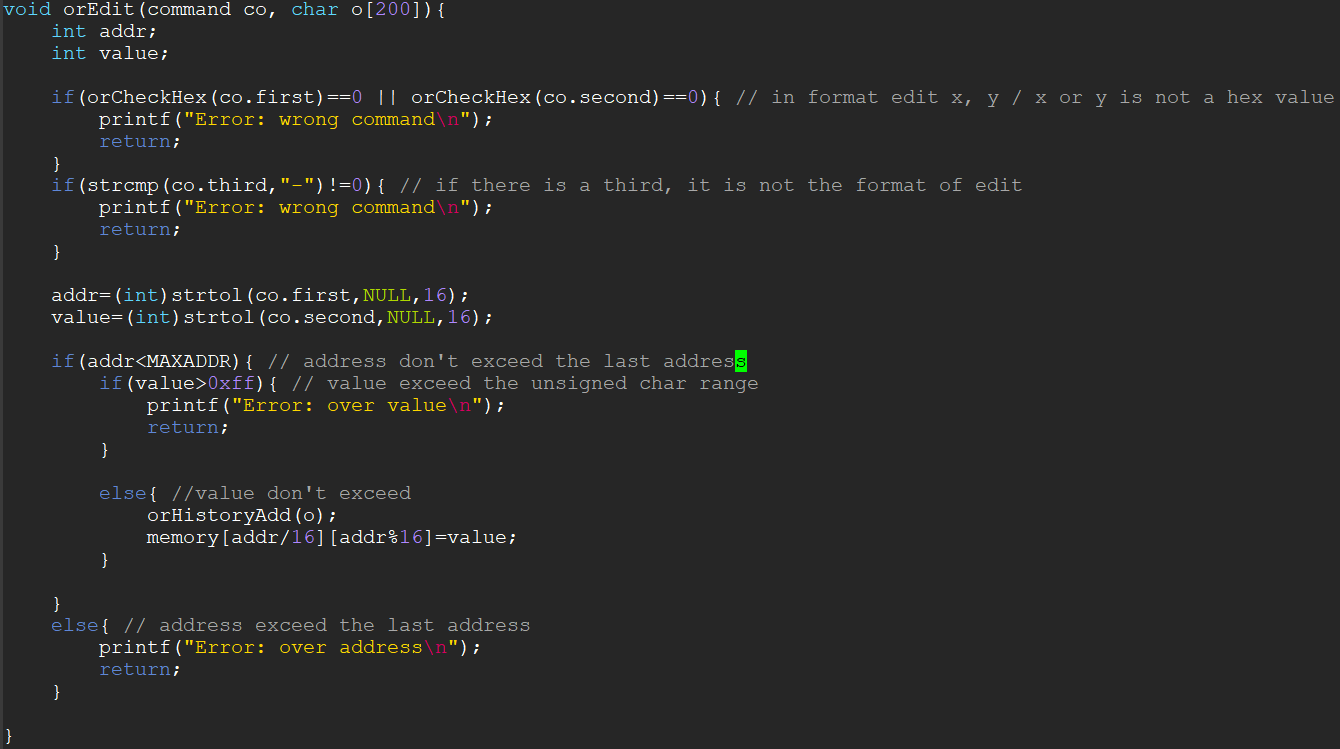
Stline에 (start/16)\*16으로 start 주소가 있는 줄의 가장 처음 주소를 넣고 edline에 (end/16)\*16으로 end 주소가 있는 줄의 가장 처음 주소를 넣습니다. Valuepoint와 charpoint는 현재 출력중인 주소를 넣을 것이므로 처음 값인 stline을 넣습니다. 이후 stline부터 edline까지 돌면서 먼저 주소를 출력해주고, 값을 차례대로 출력합니다. 이 때, 만약 valuepont가 start보다 작거나 end보다 크다면 출력하지 말아야 하는 범위이므로 스페이스바만 출력해줍니다. 아니라면 그 때의 메모리 값을 출력해줍니다. 이후 valuepoint를 +1해주어 계속 현재 주소를 갖도록 갱신합니다. 메모리값을 다 출력했다면 메모리값의 문자형을 출력해줍니다. 이 때에도 charpoint가 start보다 작거나 end보다 크다면 출력하지 말아야 하는 범위이므로 “.”를 출력해주고 아니라면 메모리 값의 문자를 출력합니다. 이 때, 메모리 값의 문자가 0x20보다 작고 0x7E보다 크다면 “.”만 출력해주고 아니라면 메모리 값의 문자를 출력합니다. 이후 charpoint를 +1해주어 계속 현재 주소를 갖도록 갱신합니다. 출력이 끝났으면 함수를 종료합니다.

* 1. **orCheckHex**



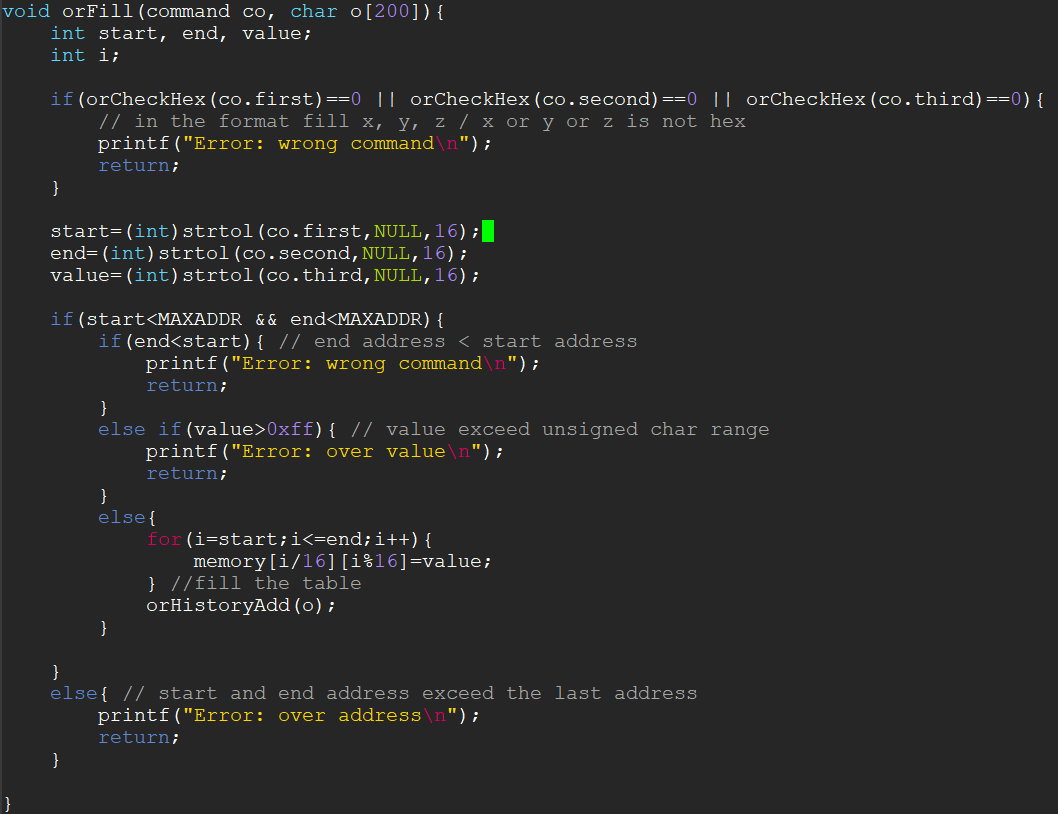
입력받은 문자열을 처음부터 ‘\0’으로 문자열의 끝이 나올 때까지 순회합니다. 돌면서 만약 문자의 값이 0~9, A~Z의 범위를 넘어간다면 hex 범위가 아니므로 0을 반환해주고 끝까지 돌았다면 문자열의 모든 문자가 hex 범위인 것이므로 1을 반환해줍니다.

* 1. **orEdit**



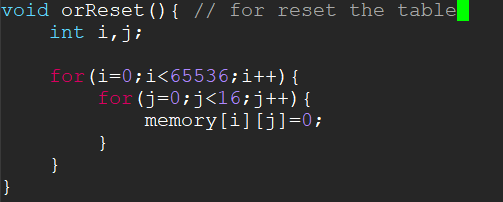
먼저 addr과 value값인 co.first와 co.second가 hex값인지 판별해줍니다. 둘 중 하나라도 hex 값이 아니라면 에러 메시지를 출력하고 함수를 종료합니다. 이후 세번째 인자인 co.third의 값이 “-“인지 확인해줍니다. 초기화 값이 아니라면 다른 값이 들어왔단 것이므로 형식에 맞지 않아 에러 메시지를 출력하고 함수를 종료합니다. 검사를 마쳤다면 addr과 value에 10진수 integer로 변환된 값을 넣어줍니다. Addr이 MAXADDR을 넘어가거나 value값이 0xff보다 크다면 잘못된 입력이므로 에러 메시지와 함께 종료합니다. 아니라면 orHistoryAdd를 불러 history에 추가하고 메모리 값을 value로 바꿔줍니다.

* 1. **orFill**



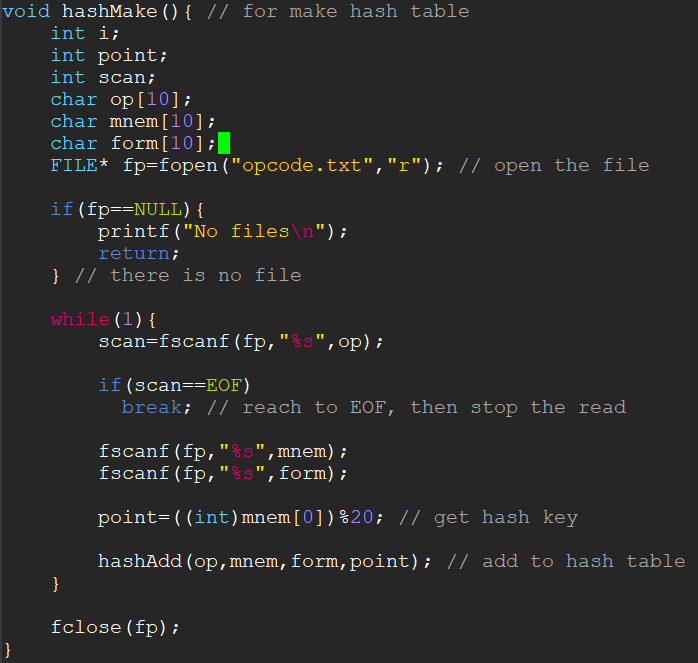
Start, end, value 값인 co.first, co.second, co.third값이 hex 값인지 판별해줍니다. 만약 하나라도 hex가 아니라면 잘못된 입력이므로 에러 메시지를 출력하고 함수를 종료합니다. Start, end, value에 각각 10진수 integer로 변환된 값을 넣고, start나 end가 MAXADDR을 넘거나 value가 0xff를 넘는 경우 잘못된 입력이므로 또한 에러 메시지를 출력하고 함수를 종료합니다. 모두 정상적으로 들어왔다면, start부터 end까지 memory를 돌며 memory 값을 value로 바꾸어주고 orHistoryAdd로 history에 추가합니다.

* 1. **orReset**



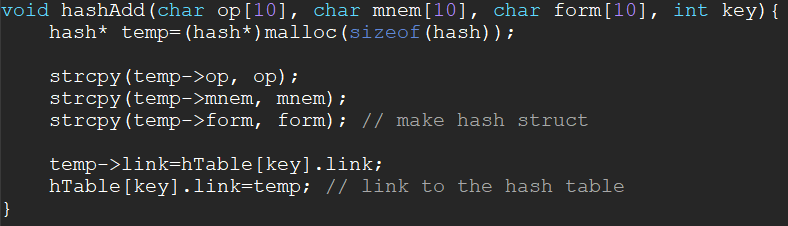
메모리 배열을 전부 돌면서 모든 값을을 0으로 초기화해줍니다.

* 1. **hashMake**



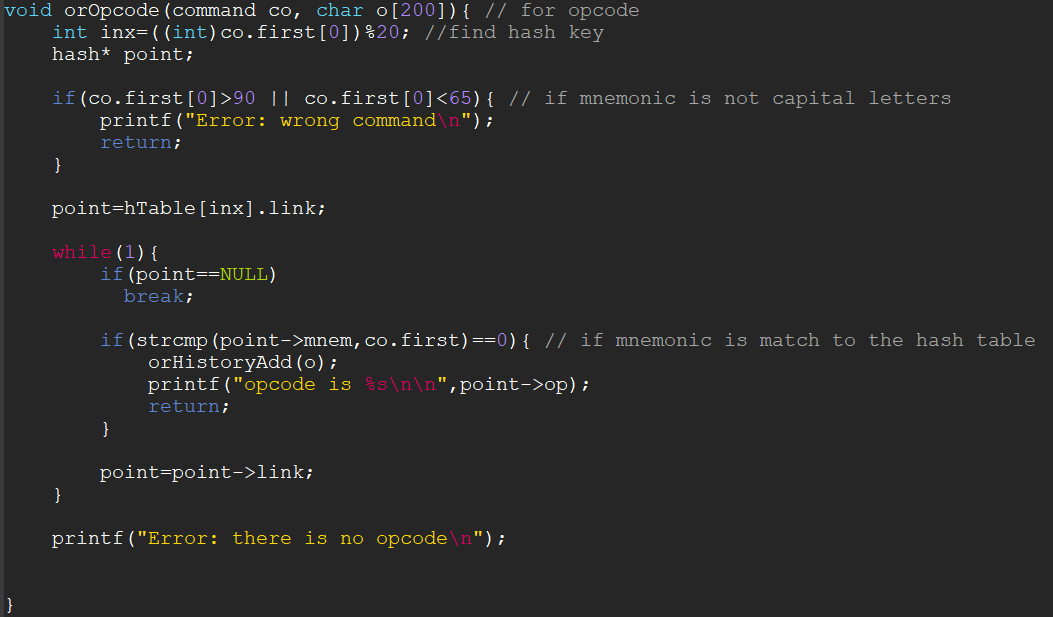
Opcode.txt를 열어 순서대로 문자열을 스캔합니다. 만약 스캔했을 때 EOF가 뜬다면 파일의 끝이므로 그만 스캔하고 함수를 종료합니다. 아니라면 3개의 문자열, opcode, mnemonic, format을 받아 각각 op, mnem, form에 넣어줍니다. Hash key는 mnem의 첫번째 문자를 20으로 나누었을 때의 나머지로 정하여 index값인 point에 넣어주고 hashAdd에 인자를 알맞게 넣어 hash table에 추가해줍니다.

* 1. **hashAdd**



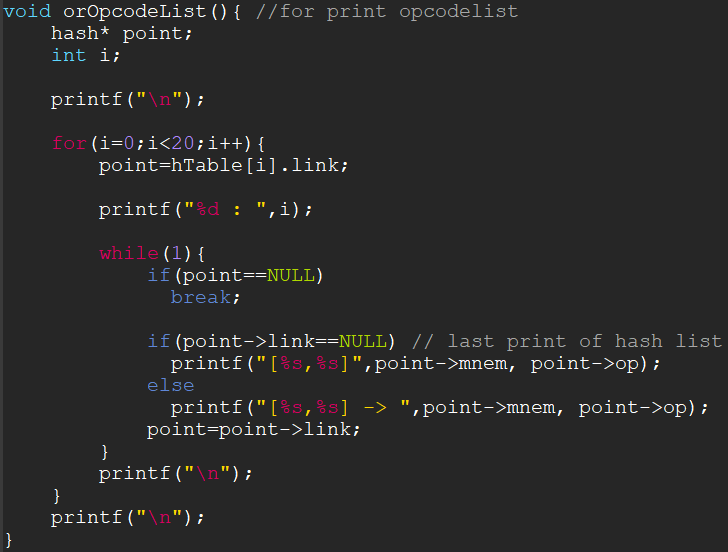
Opcode, mnemonic, format, hash key를 인자로 받아 새로운 hash 구조체 temp안에 각각 넣어주고 hash key에 맞는 인덱스로 hTable에 추가해줍니다.

* 1. **orOpcode**



Mnemonic이 저장되어 있는 co.first의 첫번째 문자를 20으로 나눈 나머지로 index를 찾습니다. 만약 co.first가 A~Z의 범위가 아니라면 에러 메시지를 출력하고 함수를 종료합니다. 맞다면 point를 hTable[inx].link로 초기화하고 hTable에 저장된 point->mnem과 co.first가 같은지 확인합니다. 같다면 opcode를 찾은 것이므로 opcode 를 출력해주고 orHistoryAdd로 history에 추가합니다. 아니라면 point가 NULL이 될 때 까지 찾은 후 못찾았다면 에러 메시지를 출력해줍니다.

* 1. **orOpcodeList**



Index 0~19까지 저장된 모든 hash table을 출력해줍니다. 이 때 현재 위치를 나타내는 point의 link가 NULL이라면 지금 출력이 이 인덱스의 마지막이라는 소리이므로 ‘->’는 출력하지 않습니다. Point가 NULL이 될 때 까지 출력하고 다음 인덱스로 넘어갑니다.